arXiv:2603.20107v1 [cs.CR] 20 Mar 2026

分布式隐私保护监控

Mahyar Karimi[1], K. S. Thejaswini[2],Roderick Bloem[3],与 Thomas A. Henzinger[1]

[1] ISTA，奥地利克洛斯特新堡 {mahyar.karimi,tah}@ista.ac.at[2] 比利时布鲁塞尔自由大学 thejaswini.raghavan@ulb.be[3] 奥地利格拉茨工业大学 roderick.bloem@tugraz.at

摘要。 在传统的运行时验证中，通常由单体监控器观察系统。在这类场景中实施隐私保护的计算成本很高，因为这需要使用复杂的密码学原语。因此，隐私保护监控仍不适用于实时应用。

在这项工作中，我们通过将监控器分布到多个参与方（其中至少一方是诚实的），来应对这一可扩展性挑战。这种架构使我们能够使用高效的秘密共享方案，而非计算成本高昂的密码学方法，从而大幅降低开销，同时维持强有力的隐私保证。现有的秘密共享方法通常仅适用于不维护内部状态的一次性执行；与此不同，我们提出了一种适用于持续监控的协议，支持针对不断变化的内部状态进行重复评估（该状态对系统和监控实体保密）。我们使用 MP-SPDZ 框架实现了这一方法。实验表明，在这些架构假设下，我们的协议比现有替代方案具有显著更强的可扩展性。

关键词： 隐私保护运行时验证 · 秘密共享。

1 引言

如今，许多软件都是专有软件，包括医疗保健、银行、地铁系统、生物信息学软件、恶意软件检测系统，甚至验证软件。由于验证软件和软件中待验证的数据双方都需要隐私保护，因此第三方验证难以实施。如果软件的受保护信息或其处理的敏感数据遭到泄露，隐私性就会受到损害。此外，泄露待验证的规范也可能破坏第三方验证的本质。例如，银行需要在不泄露交易算法的情况下向审计员证明其符合法规要求；医疗器械制造商则需要在保护控制逻辑的同时寻求安全认证。这些例子凸显了一项根本挑战：如何在保护敏感信息机密性的同时，验证其是否符合形式化规约。

与需要完整模型描述才能确定是否符合规范的静态验证不同，运行时验证只监控正在运行的系统所产生的特定输出。通过避免分析整个系统状态的需要，运行时监控为隐私保护验证提供了一种更具可扩展性的方法 [23,24,25,24]。在大多数运行时验证场景中，隐私要求是双向的。系统包含必须对验证者保密的敏感数据（客户信息、商业秘密），而披露 φ 则可能损害独立验证。

相关工作。 近期的隐私保护监控方法主要依赖密码协议，以确保生成可观测输出轨迹的系统与持有规范（例如确定性有限自动机或STL公式）的监控器之间的交互安全。所有这些工作都考虑了隐私保护设置：监控器无法获知系统的轨迹，同时系统也无法获知规范。

Banno 等人 [1] 提出了使用全同态加密（FHE）进行隐私保护监控的方法，使监控器能够私密地验证线性时序逻辑（LTL）规范。Waga等人 [29] 随后将该方法扩展至信号时序逻辑（STL）。在这些基于FHE的框架中，由于FHE协议的特性，监控结果仅向系统公开，监控器无法对其进行验证。此外，Banno 等人和Waga等人能够抵御恶意对手，而下面讨论的大多数其他工作则假设对手是半诚实的（诚实但好奇）。

另外，Henzinger、Karimi 和 Thejaswini [16] 采用混淆电路（GC）和私有函数求值。与基于全同态加密的工作有一项关键区别：在这种设置下，轨迹是否满足系统的要求，只会向监控器揭示。在近期一项独立工作中，Koll、Hang、Rosulek 和 Abbas [21] 也将 GC 协议用于监控信息物理系统的 STL 规范。然而，Koll 等人要求预先固定轨迹长度，因此所需时间会随预设的轨迹长度上限线性增长。相比之下，Henzinger 等人通过提供一种新的混淆电路改进方案，支持在运行时确定可变的轨迹长度。

尽管这些解决方案提供了强有力的隐私保证，但计算开销巨大：由于底层加密协议的复杂性，监控延迟往往导致验证即使是简单的规范也需要数十分钟，因此难以用于实时监控。

我们的方法。 我们研究隐私保护监控问题：系统生成可观察的输出（例如，银行客户的一系列交易），由监控器依据规范进行验证。我们考虑一种一般性场景，其中规范可能具有时间性质，并且依赖于不断变化的内部状态（例如银行余额或特定客户的交易）。

我们实施双向隐私：监控器只能获知规范是得到满足还是遭到违反，而不会获知系统的输出轨迹

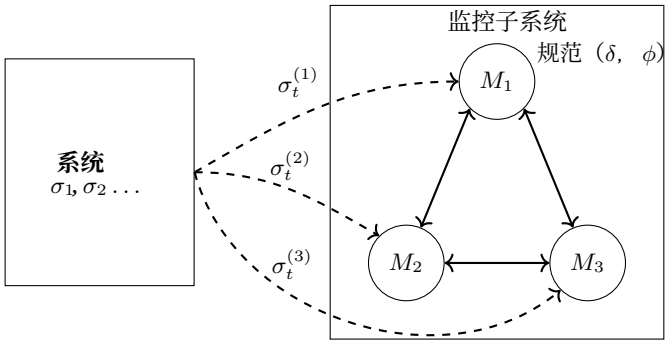

图 1：分布在多个监控器上的监控实体。

以及内部规范状态；反过来，系统只能获知评估结果，而无法了解规范本身。

与依赖繁重密码学原语的以往工作不同，我们利用底层架构来实现隐私性和效率。我们将监控器建模为一组分布式参与方，而非一个整体实体，其中至少有一台计算机是诚实的。这使我们能够利用秘密共享来监控私有数据上的秘密规范，具体原理将在下文说明。虽然标准的秘密共享方案通常仅适用于无状态的一次性函数，但我们考虑了一种用于监控的有状态 协议。这样便能在维护持久且私密的内部状态的同时，反复评估规范。

两种共享的故事。 设想系统持有一个秘密字节 $\sigma \in \{0, \ldots, 255\}$，并希望将其分发给三台监控计算机。它只需将 σ 切分为三个随机份额：随机选取 $\sigma^{(1)}$， $\sigma^{(2)}$，计算出满足 $\sigma^{(1)} + \sigma^{(2)} + \sigma^{(3)} \equiv \sigma \pmod{256}$ 的 $\sigma^{(3)}$，然后将每个份额交给一台计算机。单独看，每个份额都像纯粹的噪声；即使两台计算机串通，它们也缺少拼图中的第三块。只有三台计算机合并各自的份额，秘密 σ 才会显现。如果系统生成一个输出流（x_t， y_t），而监控器需要跟踪类似 $\sum(5x_t + 8y_t)$ 的加权运行和，计算机便可以直接利用各自的私有份额在本地执行计算。只要有一台计算机是诚实的，各个数值就仍然保密；三台监控器只能获知加权和，而无法获知各个数值。

假设规范要求将数值相乘（$x \cdot y$）。在这种情况下，加法方法行不通，因此我们改用 Shamir 的多项式秘密共享方法 [28]。系统通过选择两个随机二次多项式（ p 和 q），来隐藏秘密 x 和 y，使得 $p(0) = x$ 且 $q(0) = y$。系统向每台计算机提供每条曲线上的一个点：计算机 i 获得 $(i, p(i))$ 和 $(i, q(i))$。由于三个点可以唯一确定一条抛物线，因此三台计算机（但少于三台则不行）可以联合恢复这些秘密。

问题出在尝试相乘时。每台计算机都可以在本地将其 y 坐标相乘，从而找到乘积曲线 $r(z) = p(z) \cdot q(z)$ 上的一个点。

然而，两条二次曲线相乘会得到一条四次曲线（次数为 4）。定义一条四次曲线需要五个点，但我们只有三台计算机！本地份额已不足以重构结果。要解决这个问题，需要额外的交互步骤（例如次数约简），这是标准做法。

现实世界中的规范很少只是求和或只是求积。验证诸如 $\sum(x_t^2 + y_t) < 100$ 这样的条件，需要用多项式秘密共享处理乘积和求和，用布尔秘密共享处理比较。尽管针对各个领域都有专门的方案，但它们依赖彼此不兼容的数据表示形式：不能直接将算术份额输入布尔电路。

像 MP-SPDZ 这样的现代多方安全计算框架 [18]提供了在这些表示形式之间进行转换的机制，但它们通常面向有限的一次性计算。它们不支持运行时监控“无限”、持续运行的特性；在运行时监控中，必须跨事件流持续维护并不断演进秘密状态。

我们的协议。 在我们的协议中，系统的输出轨迹通过秘密共享分布在各个监控器参与方之间。我们在图 1中以图示方式展示了这一点。初始化后，系统与监控器每个观测步骤只需交换一条消息即可评估规范，从而免去了昂贵的密码学运算。与全同态加密或混淆电路相比，这带来了数量级的加速。其核心思想是，尽管整个监控子系统“知晓”系统可观测输出的秘密共享版本，但监控子系统中任何真子集的计算机都无法知晓系统的可观测输出。

为了能够使用不同的份额格式并执行各种布尔型和算术型运算，我们的协议结合了多种秘密共享技术。我们采用份额转换协议，使监控方能够在不同表示形式之间切换：例如，将算术份额（适合高效求和）转换为布尔份额（适合高效比较），而始终不会泄露底层数值。重要的是，这使我们能够在重复评估过程中维持持久且私密的执行状态，从而支持需要混合使用算术运算和逻辑运算的复杂规范。借此，我们监控的内容不再局限于不变量，还能涵盖有状态协议。

这种性能表现依赖于特定假设：分布式监控方中必须有诚实多数。我们证明，在这些约束下，该协议能够维持强信息论隐私：任何单个参与方（或不诚实的合谋方）都无法获知秘密数据，各实体也无法了解规范违规序列之外的任何信息。

除了协议之外，我们还提供了一个基于线性秘密共享方案的安全多方计算框架，该框架本身也具有独立的研究价值。尽管 Cramer、Damgård 和 Maurer [8] 也提出了类似的抽象，但我们的框架将适用范围从线性情形推广到了更广泛的一类共享系统，使我们能够对在不同秘密共享表示之间切换的协议进行推理。

我们在四种不同的监控场景中评估了我们的方法：访问控制系统、分布式锁管理、血糖监测和地理围栏。这些案例研究代表的是有状态的时间规范，而非简单的

无状态不变量。我们采用了 Henzinger 等人的前两项基准测试，[16]以及 Banno 等人的第三项基准测试。[1]此外，我们还引入了一项新的地理围栏基准测试，以展示我们的方法高效处理非线性运算的能力。与以往的密码学方法相比，我们的实现速度快了数个数量级；在大多数场景中，每次迭代的监控耗时远低于一秒。这使得实用的实时监控成为可能，同时还能提供信息论隐私保证。

2 问题定义

我们对问题进行正式定义，并讨论本文使用的记号。这里有两个实体，分别是系统（有时也称为 $\mathcal{S}$）和监控器 $\mathcal{M}$。监控实体至少由两台计算机组成，而系统被视为单体，由一台计算机组成。我们用 M_i 表示监控实体中的 i^{th} 计算机。对于任意一对计算机（C，D），无论它们属于哪个实体，C 和 D之间都至少存在一条私有通信信道。

系统。 系统是一个有限状态机，其状态 σ 是一个 n比特的二进制字符串。系统生成无限状态序列 σ_1，σ_2，$\cdots \in \{0,1\}^n$，其中 σ_t 是在时间 t生成的。系统产生的这一无限序列也是可观测输出。

监控器。 监控器以一个规范作为输入，该规范的形式是一个有限状态机，其状态空间为 $\{0,1\}^m$，并具有两个函数：下一状态函数δ 和标志函数 ϕ。监控实体中的所有计算机都知道 δ 和 ϕ。

– 在时间点 t，下一状态函数 δ 接收输入 $(\mu_t,\ \sigma_t)$，与 ϕ 的输入类似，并输出下一个规范状态 $\mu_{t+1} = \delta(\mu_t,\ \sigma_t)$。– 标志函数 $\phi(\mu,\ \sigma)$ 以当前规范状态 $\mu \in \{0,1\}^m$ 和系统输出 σ为输入，并返回一个布尔值。当且仅当 $(\mu,\ \sigma)$ 是非期望输入时，我们记作 $\phi(\mu,\ \sigma) = \top$。

协议。 协议 P 是由通信顺序进程组成的集合。对于每台计算机 X（无论属于系统还是监控器），协议都会分配一个非空进程 $p(X)$，该进程在 X 上运行。整个协议期间，所有计算机都会执行分配给它们的进程。

正确性。 协议 P 当且仅当在每个时间点 t，监控实体的计算机能够共同获得 μ_{t+1} 和 $\phi(\mu_t,\ \sigma_t)$ 的正确值时，才是正确的。这并不要求该实体中的任何一台计算机都必须持有正确值；正确性仅要求该实体能够汇集其计算机本地可用的信息来计算这些值。如果协议 P 是正确的，那么它将在满足 $\phi(\mu_t,\ \sigma_t) = \top$ 的第一个时间点 t 终止。

隐私性。 协议 P 当且仅当在每个时间步 t，每个实体（系统和监控器）仅 获知评估结果 $\phi(\mu_t、\ \sigma_t)$，而不会获知有关另一实体的内部状态或转换的任何信息（超出该输出在逻辑上所蕴含的信息）时，才是保护隐私的 。

特别是，系统不得获取有关监控器状态的信息 μ_t 或转移函数的信息 δ，监控器也不得获知系统的输出 σ_t 或其超出评估结果范围的转移逻辑。形式上，隐私性通过不可区分性来定义：对于控制任一实体的任何对手，协议执行中涉及另一实体秘密的视图都应无法与一串均匀随机值 [22]区分。

重复执行。如果协议支持针对不断演变的序列 $(\sigma_t)_{t\geq 1}$ 进行重复执行，且每次执行的计算开销都不大，则该协议满足可行的重复执行。

假设。我们假设任意两台计算机之间的通信都是私密的：对于每一对计算机 $(P,\ Q)$，P 可以向 Q 发送消息，使得除 P 和 Q 之外的任何计算机都无法获知该消息的任何信息。

安全模型。我们采用半诚实腐化模型。所有计算机，包括被腐化的计算机，都会严格按照协议规定执行，但被腐化的计算机可能会汇集其内部存储，试图推断出更多信息。在此模型下，安全性保证这类被腐化的计算机集合无法获知超出其自身输入和输出已揭示内容之外的信息，尤其不会泄露有关另一实体内部状态的非预期信息。

问题陈述。 给定系统和监控实体、它们的内部状态机，以及上一节定义的可观测输出 σ_t，我们的目标是设计一种协议 P，使这两个互不信任的实体能够联合评估规范，同时不泄露任何额外的内部信息。

在每个时间点 t，监控器必须获得 $\mu_{t+1} = \delta(\mu_t,\ \sigma_t)$ 的正确值以及标志 $\phi(\mu_t,\ \sigma_t)$，同时系统和监控器不得获知标志本身的值以外的任何信息。这两个实体都由多台计算机组成，并在对手行为的半诚实模型下运行。

我们寻求一种协议 P，使其满足正确性和隐私性，并支持可行的重复执行。

3 秘密共享

隐私保护监控的核心挑战是在不泄露底层数据的情况下实现验证。秘密共享提供了一种解决方案：我们不将某个值存储在一个位置，而是以一种方式将其分散到多台计算机上，使得任何一台计算机都无法获知该值，但所有计算机共同协作时却能够重构该值并对其进行计算。

对份额进行计算。 秘密共享的真正优势在于，我们可以对共享值进行计算而无需重构它们。例如，假设每台计算机都持有两个秘密比特 a 和 b的一个份额，分别记作 $[\![a]\!]$ 和 $[\![b]\!]$。为了在确保所有内容保密的同时计算异或 $c = a \oplus b$，每台计算机 i 只需在本地计算$c^{(i)} = a^{(i)} \oplus b^{(i)}$ 。由于

$$\begin{aligned} c^{(1)} \oplus c^{(2)} \oplus c^{(3)} &= (a^{(1)} \oplus b^{(1)}) \oplus (a^{(2)} \oplus b^{(2)}) \oplus (a^{(3)} \oplus b^{(3)}) \\ &= (a^{(1)} \oplus a^{(2)} \oplus a^{(3)}) \oplus (b^{(1)} \oplus b^{(2)} \oplus b^{(3)}) = a \oplus b = c, \end{aligned}$$

这些计算机现在持有 c 的有效共享，却从未获知 a、 b 或 c。

计算若干比特的与运算更加复杂：计算机必须进行交互并使用额外的随机性来防止信息泄露。这种不对称性——有些操作在本地完成，另一些则需要交互——是安全多方计算的基础，也促使我们需要采用混合协议计算。

我们通过抽象的秘密共享概念来描述我们的协议。这样，我们就能独立于任何具体的安全多方计算后端或表示形式，来描述正确性和隐私性 [14,8]。

我们的模型组织如下。首先，我们定义抽象共享系统（第 3.1 节），它概括了任何秘密共享方案必须满足的基本属性。然后，我们将份额转换（第 3.2 节）引入为一种一等原语，以实现不同秘密共享表示之间的安全互操作。最后，我们给出具体的实例化（附录 A.1），并为份额转换确立效率标准。

我们将 Cramer、Damgård 和 Maurer [8] 提出的现有框架扩展到更广泛的共享系统类别；他们提供了一个基于线性秘密共享方案构建安全多方计算的框架。关键的是，我们明确将份额转换表述为一种原语操作，并为其规定形式化正确性和隐私要求（定义2）。在统一接口下以一致方式处理异构安全多方计算后端，有助于我们推理在不同秘密共享表示之间切换的协议。

3.1 抽象共享系统

抽象共享系统将秘密值与其分布式表示之间的接口形式化。核心思想是将值 v 分解为份额，并将其分发给 k 台计算机，使得任何真子集的计算机都无法获知关于 v 的任何信息，而所有计算机协同起来则能够重构 v 并对其计算函数。

定义 1（抽象共享系统）。一个抽象共享系统，适用于k 台计算机，是一个元组 $\mathcal{S} = (\mathcal{V},\ \mathcal{D},\ \text{共享},\ \text{重构}\ \mathcal{O})$，其中：

– $\mathcal{V}$ 是秘密值的集合。– $\mathcal{D}$ 是份额的定义域。– 共享函数： $\mathcal{V} \to \mathcal{D}^k$ 是一个（可能随机化的）共享函数，每台计算机生成一个份额。– 重构： $\mathcal{D}^k \to \mathcal{V}$ 是一个确定性重构函数。

– $\mathcal{O}$ 是对共享值执行的支持运算集合，其中每个 $f \in \mathcal{O}$都满足 $f \in (\mathcal{V}^r \to \mathcal{V})$，对于某个 $r \in \mathbb{N}$。

每台计算机 $i \in \{1, \ldots, k\}$ 持有 Share(v) 的第 i 个分量，记作 Share(v)$_i \in \mathcal{D}$。共享函数可能是概率性的，因此 Share(v) 表示 $\mathcal{D}^k$ 上的一个随机变量。

共享系统必须满足以下性质。

性质 *1*（正确性）。对于所有 $v \in \mathcal{V}$，Reconstruct(Share(v)) $= v$ 都以概率 1 成立，其中概率是针对 Share 的随机性而言。

性质 *2*（隐私性）。对于计算机的任意真子集 $I \subset \{1, \ldots, k\}$，份额的联合分布 $\{\text{Share}(v)_i\}_{i \in I}$ 与 v 相互独立。形式化地，对于所有 $v,\ v' \in \mathcal{V}$, $\{\text{Share}(v)_i\}_{i \in I} \equiv \{\text{Share}(v')_i\}_{i \in I}$，其中 $\equiv$表示分布相等。

这一性质确保了完美隐私：即使多达 $k-1$ 台计算机组成的合谋方，也无法获知任何超出其原本已知范围的秘密。

性质 *3*（运算封闭性）。对于每个元数为 r 的运算 $f \in \mathcal{O}$，都存在一个协议 Π_f，该协议接收值 v_1，…，$v_r \in \mathcal{V}$ 的共享 $[\![v_1]\!]$，…，$[\![v_r]\!]$，生成一个共享 $[\![f(v_1, \ldots, v_r)]\!]$，且不会向任何计算机泄露任何中间值。

协议 Π_f 可以涉及交互和随机性，但必须保持隐私性：任何计算机的真子集都无法获知关于v_1，…，v_r或 $f(v_1, \ldots, v_r)$的任何信息。

记号。我们用 $[\![v]\!]_{\mathcal{S}}$ 表示在系统 $\mathcal{S}$ 下对 v 的共享；当上下文明确时，省略下标。当我们写作 $[\![v]\!]$时，指的是一个份额元组 $(s_1, \ldots, s_k) \in \mathcal{D}^k$，满足 Reconstruct($s_1$，…，$s_k$) $= v$。

3.2 份额转换

在实际应用中，计算的不同部分可能需要不同的共享表示形式。例如，比较操作通常采用布尔形式会更高效，而算术运算则适合采用加法共享或多项式共享。份额转换使协议能够在不同表示形式之间切换，而无需重构底层秘密。设

$\mathcal{S}_1 = (\mathcal{V}_1,\ \mathcal{D}_1,\ \text{份额}_1, \text{重构}_1,\ \mathcal{O}_1)$　$\mathcal{S}_2 = (\mathcal{V}_2,\ \mathcal{D}_2,\ \text{份额}_2, \text{重构}_2,\ \mathcal{O}_2)$

是两个共享系统。

定义 2（份额转换）。从 $\mathcal{S}_1$ 到 $\mathcal{S}_2$ 的份额转换是一个协议$\text{Convert}_{\mathcal{S}_1 \to \mathcal{S}_2}$：$\mathcal{D}_1^k \to \mathcal{D}_2^k$，使得对于每个 $v \in \mathcal{V}_1 \cap \mathcal{V}_2$:

1. 正确性：转换后重构得到相同的值：重构$_2$(Convert(份额$_1(v)$)) $= v$，其概率为1，该概率是针对份额$_1$ 和Convert的随机性而言。

2. 隐私性： 该协议不会泄露关于 v 的任何信息，除输出共享所能推知的信息外。形式上，对于任意真子集 $I \subset \{1,\ldots,k\}$ 以及任意v、 $v' \in \mathcal{V}_1 \cap \mathcal{V}_2$，计算机 I 在Convert 执行期间的视图，与其在输入份额$_1(v)$ 时的视图不可区分。

我们将份额转换视为一等原语。其内部实现可能涉及交互、随机性或辅助共享值，但这些细节均被抽象化。

备注 *1*（组合性）。份额转换可以自然地进行组合。也就是说，如果 Convert $_{\mathcal{S}_1\to\mathcal{S}_2}$ 和 Convert$_{\mathcal{S}_2\to\mathcal{S}_3}$ 是份额转换，那么它们的组合 Convert$_{\mathcal{S}_2\to\mathcal{S}_3}$ ∘Convert $_{\mathcal{S}_1\to\mathcal{S}_2}$ 就是一个从 $\mathcal{S}_1$ 到 $\mathcal{S}_3$ 的份额转换。这直接源于正确性和隐私性属性。

在附录 A中，我们使用标准的安全多方计算方案，为抽象共享系统提供具体实例化：环上的加法共享 $\mathbb{Z}_{2^k}$ 用于算术计算，多项式（*Shamir*）共享用于门限设置下的信息论安全，以及布尔共享用于逻辑谓词。

为支持比较（$x < y$）和相等性检查等非线性运算，我们使用高效份额转换协议，其中最重要的是比特分解，它将算术份额转换为各个比特的布尔份额。线性运算可以本地高效完成，而非线性运算则需要交互：乘法利用预计算的 *Beaver* 三元组来实现常数轮复杂度，而比较则依赖份额转换协议，随后使用优化的布尔电路。

混合协议计算。 现代安全多方计算框架利用不同表示形式在效率上的差异：算术运算（加法、乘法）在 O 轮中基于加法份额或 Shamir 份额进行，布尔运算（与、或、异或）在 O 轮中基于布尔份额进行，而仅在跨越表示边界时才执行转换 [13,26]。通过谨慎选择表示形式以尽量降低转换开销，混合协议计算的具体效率显著高于始终使用单一表示形式的方案 [26]。

我们作出以下假设：协议所需的所有份额转换，均可按照定义 4 中的规定高效实现。

4 分布式监控协议

在建立抽象共享框架之后，我们现在介绍如何利用该框架构建隐私保护监控协议。关键挑战在于，使监控器能够根据系统的输出评估规范，却无法获知输出本身；同时确保系统无法预测或影响监控器所获知的信息。

本节安排如下。首先，我们介绍设置阶段和状态表示（第 4.1 节）。然后，我们详细说明逐轮执行过程（第 4.2 节），并形式化定义确保隐私性和正确性的属性（第 4.3 节）。

4.1 设置与状态表示

执行开始前，系统和监控器会建立秘密共享的通用框架，并商定规范的表示方式。

共享参数。系统与监控器就一个固定的共享系统达成一致$\mathcal{S} = (\mathcal{V},\ \mathcal{D},$ 共享，重构 $\mathcal{O})$。该约定包含对元组中每个元素的具体定义；例如，值域可以是模 2^{64} 整数或比特$\mathcal{V}$，支持的运算集合可以是算术型或逻辑运算，或者比较 $\mathcal{O}$。

协议期间交换的所有共享都是 $\mathcal{S}$ 的实例。这确保系统共享的值可由监控器直接用于本地计算，无需在不兼容的表示形式之间进行额外转换。请注意，监控计算机之间也可以使用其他约定好的共享系统来进行份额转换。

持久共享状态。监控实体在整个执行过程中都以共享形式维护规范状态μ_t 。在每个时间点 t，以下不变量均成立：

> 监控计算机共同持有一个在秘密共享系统 $\mathcal{S}$ 下的份额 $[\![\mu_t]\!]$，且没有任何单个监控计算机能够获知 μ_t。

这是核心隐私机制：尽管监控器们共同知晓当前规范状态，但没有任何一台监控计算机能够在未经其他计算机协助的情况下提取该状态。

相比之下，系统会在本地维护其内部状态 σ_t，不会与监控器共享该状态。系统的状态可能包含敏感信息，例如密码密钥、用户数据或专有算法。该协议确保此状态完全保密。

4.2 逐轮执行

执行以离散轮次进行，轮次由 $t = 1$、2、..... 编号。每一轮都会将共享的监控器状态从 $[\![\mu_t]\!]$ 转换为 $[\![\mu_{t+1}]\!]$，并生成一个违规标志比特。

第 t 轮由三个阶段组成。

1. **共享可观测输出**。系统根据约定的共享方案 $\mathcal{S}$，计算共享值 $[\![\sigma_t]\!] = \mathrm{Share}(\sigma_t)$。系统通过私有信道将 $[\![\sigma_t]\!]$ 的 i^{th} 份额发送给 i^{th} 监控计算机。

   > 完成此步骤后，监控实体共同持有 σ_t的一个共享，而任何单个监控计算机都无法获知 σ_t 除其自身份额之外的任何信息。根据性质2，任何监控计算机的真子集都无法获知 σ_t。

2. **规范评估**。监控计算机使用共享输入 $[\![\mu_t]\!]$（来自上一轮）和 $[\![\sigma_t]\!]$（来自当前轮次），共同评估该规范。

   更准确地说，它们执行安全多方计算，生成：– 共享的违规标志 $[\![\phi(\mu_t, \sigma_t)]\!] \in [\![\{0,1\}]\!]$，以及
   – 下一个规范状态 $[\![\mu_{t+1}]\!] = [\![\delta(\mu_t, \sigma_t)]\!]$ 的共享编码。
   此计算完全通过 $\mathcal{S}$ 支持的操作和转换来表达。例如，计算比较 $\sigma_t > c$ 可能需要将 $[\![\sigma_t]\!]$ 从算术型转换为布尔形式、执行按位比较，再将结果转换回来。所有中间值始终保持秘密共享状态——绝不会重构任何中间结果。

3. **标志重构与控制流**。监控计算机使用重构函数共同重构共享比特 $[\![\phi(\mu_t, \sigma_t)]\!]$。这是每轮中唯一被重构的值。

   如果重构出的值为 $\top$（表示发生违规），协议就会终止，并向外部观察者报告违规。否则，重构出的标志会被丢弃，监控器将 $[\![\mu_{t+1}]\!]$ 保留为下一轮的持久共享状态。规范状态 μ_{t+1} 永远不会被揭示。

4.3 不变量与信息流

现在，我们将形式化描述在整个执行过程中始终成立的关键性质。

性质 *4*（状态演化不变量）。在每轮 t 开始时，监控实体共同持有 $[\![\mu_t]\!]$，且不会以明文形式存储有关执行历史的其他信息。协议执行期间唯一会被重构的值，是每轮的违规标志 $\phi(\mu_1,\ \sigma_1)\ \phi(\mu_2,\ \sigma_2)$.....。

这一不变量确保协议的信息泄漏最小且明确：每轮泄漏一比特，仅此而已。

4.4 计算模型

为使规范评估阶段更加具体，我们描述监控器评估规范所依托的计算基础。关键挑战在于明确追踪份额转换发生的位置，并确保所有操作对于共享表示而言都是类型安全的。

我们采用一种带有基于寄存器的内存的类型化指令集架构，类似于 MP-SPDZ 等实用安全多方计算系统所使用的方法 [17]。尽管这种设计受到现有多方安全计算框架的启发，我们仍在附录 B中对其进行形式化，并为我们的协议提供精确的操作语义，以便在分析中明确份额转换的成本。

为了对隐私保护监控的执行过程建模，我们将监控器形式化为一台抽象机，它在一组有限的类型化寄存器上运行，其中每个寄存器 R_i 都保存着由分布式参与方秘密共享的值。我们定义了一套特定的指令集，用于操作这些寄存器，而不重构

秘密，并强制遵循严格的类型规则：算术运算（ADD、MUL）作用于算术份额，布尔运算（AND、XOR）作用于布尔份额，而比较操作（LT、EQ）则充当二者之间的桥梁。比较指令会在内部执行开销较大的份额转换协议，以将算术输入转换为布尔输出。通过这一抽象，我们可以将高级规范编译为类型正确的指令序列。此外，它还有助于我们直接计算份额转换的计算成本。

性质 *5*（从系统到监控器的信息流）。从系统到监控器的所有信息流都仅通过对可观测输出 σ_t 进行新鲜共享来实现。更明确地说，系统绝不会接收到源自 $[\![\mu_t]\!]$ 或监控器中间计算的信息。

此性质确保系统无法获知监控器正在检查什么，也无法根据规范状态调整其行为。

性质 *6*（从监控器到外部的信息流）。从监控器到外部世界的所有信息流都仅通过违规标志 ϕ（μ_t、σ_t）的重构来实现。具体而言：单个监控计算机永远不会获知超出其所持有的 σ_t 份额及重构出的标志所能推知的有关 σ_t 的信息；而有关 μ_t 或中间值的信息也绝不会泄露给任何一方。

性质5 和6 共同形式化了系统隐私性与监控器隐私性之间的分离。

定理 1（跨轮次组合性）。即使无限次重复执行该协议，隐私性仍能得到保障。

协议的每一轮彼此独立，唯一的例外是监控器向后续轮次传递的共享状态 $[\![\mu_t]\!]$。由于各轮之间除了违规标志之外不会泄露任何额外信息，我们证明了即使协议无限次重复执行，隐私性仍能得到保护。这一组合性对于长期运行的系统至关重要，因为监控轮次的数量并非预先固定。

4.5 对手模型与隐私保证

回顾第 2 节，我们采用半诚实腐化模型：被腐化的计算机严格遵循协议规范，但可能汇集其内部状态以推断额外信息。我们考虑静态腐化，即在执行开始前就确定被腐化计算机的集合。

定义 3（对手结构）。对手结构是一个集合(I_M)，其中 $I_M \subseteq \{M_1, \ldots, M_{k_M}\}$ 是监控实体中被攻陷计算机的集合。我们要求 $|I_M| < k_M$（即每个实体至少有一台诚实计算机）。

控制 I_M 的对手可以观察发送至或来自 I_M 中被攻陷计算机的所有消息、被攻陷计算机执行的所有本地计算，以及被攻陷计算机使用的所有随机数。对手不得观察诚实计算机之间的通信或其内部状态。

定理 2（系统隐私）。对于任意满足 $|I_M| < k_M$ 的对手结构 (I_M)，受攻击的监控计算机的视图不会泄露系统的内部状态 σ_t 或可观测输出 x_t，除非这些信息可由重构的违规标志序列 $\phi(\mu_1, x_1) \ldots, \phi(\mu_T, x_T)$ 推知。

定理 3（监控器隐私）。对于任意满足$|I_M| < k_M$的对手结构 (I_M)，受攻击的系统计算机的视图不会泄露监控器的规范状态 μ_t、下一状态函数 δ 或标志函数 ϕ，除非这些信息可由重构的违规标志序列$\phi(\mu_1, x_1) \ldots, \phi(\mu_T, x_T)$ 推知。

推论 1（联合隐私）。在满足 $|I_M| < k_M$ 的对手结构 (I_M) 下的半诚实模型中，协议仅向对手泄露违规标志序列。

最后我们指出，协议的隐私保证取决于底层共享系统。如果共享系统提供完美（信息论）隐私——例如加法共享、多项式（Shamir）共享和布尔异或秘密共享——那么我们的协议能够实现针对计算能力无界对手的完美隐私。然而，如果共享系统或份额转换协议依赖于计算假设（例如用于布尔运算的混淆电路，或基于格的转换），那么隐私性仅能针对多项式时间有界对手得到保证。

5 实验

我们使用 MP-SPDZ 框架实现并评估了我们的隐私保护监控协议 [17]，，具体采用具备外部输入输出能力的 Shamir 秘密共享。实现文件和实验脚本可在我们分叉的 MP-SPDZ 仓库中找到，位于 `experiments` 分支。[4] 我们的实验设置展示了该方法在实时监控场景中的实际可行性。

5.1 实现细节

安全多方计算框架。 我们使用 MP-SPDZ 的 Shamir 秘密共享协议，该协议为隐私保证提供信息论安全。与计算开销较大的密码学方法不同，Shamir 秘密共享在有限域中运作，不依赖计算困难性假设。本框架中的一个模板监控程序见附录 D。

[4] https://github.com/mahykari/MP-SPDZ/tree/experiments/

实现架构。 我们的实现利用 MP-SPDZ 的客户端接口，将数据提供方（系统实体）与计算参与方（监控实体）的角色分离。数据提供方提供秘密共享输入，而不参与计算；监控计算机则执行监控协议，但不会获知各个系统的秘密。实现过程中，我们使用了 MP-SPDZ 提供的秘密共享操作原语，包括时序逻辑规范中涉及的比较和条件求值。要了解如何高效组合这些原语——尤其是阈值比较和谓词布尔组合等操作——需要熟悉该框架的域算术和份额转换方法。

安全配置。 所有实验均使用包含 3 个参与方的 128 位域，腐化阈值为 1；也就是说，任意两个监控计算机都不会汇集各自的信息。

网络安全。 MP-SPDZ 中参与方之间的通信默认通过 CryptoPlayer 组件使用 TLS 加密，从而提供机密性、身份验证和完整性。系统使用基于 X.509 证书的身份验证，并采用 TLS 1.2；每对参与方之间会建立两个独立的 TLS 连接（一个用于发送，一个用于接收），以处理双向通信。

5.2 实验结果

我们使用 Henzinger 等人的两个案例研究来评估所提方法的性能和可扩展性。 [16]: 一个带有双向门的访问控制系统（ ACS ），以及一个分布式锁管理系统。此外，我们还针对 Banno 等人提出的一项规范进行了评估。 [1], 我们还评估了一个新场景，该场景展示了一项计算量更大的规范。除 Banno 等人的规范外，所有实验均使用 128 位域；监控实体由 3 台计算机组成，并运行多次迭代以测量稳态性能。

访问控制系统（ACS）。ACS 案例研究监控一栋办公楼，楼内有两类员工（A 类和 B 类），他们通过多个外部门进出。每扇门会跟踪四项数值：A 类员工和 B 类员工的进入人数与离开人数。该规范要求，当前楼内的 A 类员工人数始终不少于 B 类员工人数。我们评估了从 10 扇门到 1000 扇门的可扩展性，而 Henzinger 等人 [16]在其原始工作中只评估到 30 扇门。

分布式锁管理。 锁管理案例研究对包含多个锁的并行程序进行建模，其中每个锁有两种状态之一（加锁或解锁）。在每一轮次，监控器都会收到针对每个锁的加锁、解锁或跳过请求。规范确保对于任何锁，lock() 或 unlock() 都不会连续调用两次。这个场景本质上更加

依赖布尔运算，远甚于 ACS，需要大量布尔运算来检查状态转换。我们评估了包含 100 至 1000 个锁的设置。

总统座车地理围栏场景。总统专车场景展示了一项计算量更大的规范。监控器会跟踪一辆汽车在 n 维整数空间（$\mathbb{Z}^n$）中的位置，并在每一轮接收位移向量。该规范要求汽车始终处于一个扩张球体内：球体半径从基础值开始，随时间线性增长，并在达到最大半径后停止增长。此场景检验了安全多方计算框架的算术能力，需要在高维空间中进行非线性算术和比较。

血糖监测。Banno 等人提出的血糖场景 [1]，会监控一个定期发送读数的血糖传感器。我们评估了规范 $\square_{[600,700]}(x \leq 200)$，该规范要求在任意长度介于 600 和 700 个时间单位的时间窗口内，血糖水平始终不超过 200。Banno 等人测量了多个规范；我们选择这一项作为代表，因为其他规范呈现出相似的特征。

5.3 性能分析

电路复杂度。 表 1 展示了四种场景的离线预处理需求。对于 ACS，整数三元组的数量随门的数量线性增长：从 10 扇门时的 4,500 个三元组增至 1000 扇门时的 400,500 个。值得注意的是，位三元组（用于布尔运算）的数量和 daBit 的数量保持不变，这表明布尔运算的电路深度与门的数量无关。这符合我们的并行监控架构的预期，因为每扇门的计算彼此独立。

由于锁系统具有大量布尔运算，其预处理需求明显高于 ACS。三元组的数量随规模线性增长，从 100 把锁时的 60,100 个增至 1000 把锁时的 600,100 个。与 ACS 不同，位三元组和 daBit 的数量也随系统规模增长，反映出锁状态检查所需的大量布尔运算。

总统座驾场景随维度增长。对于 4 维情况，我们需要 3,800 个三元组；对于 1,024 维情况，数量则增至 819,800 个。与 ACS 类似，位三元组和 daBit 的数量保持不变，因为核心计算属于算术运算，而非布尔运算。

血糖场景所需的预处理极少（仅需 2 个三元组和 8 个 daBits），这反映了该规范的简单直接，其主要涉及在滑动窗口上进行阈值比较。

性能指标。右侧的表1 展示了所有四种场景的每次迭代性能指标。

对于 ACS，无论系统规模如何，计算时间都保持非常稳定（0.024–0.037 秒），而通信开销则呈线性增长。这

表 1：电路复杂度与性能指标

	电路复杂度			性能		
场景	三元组	Bit-三元组	daBits	计算 (s)	总计时间（秒）	Data (MB)
ACS（门）						
10扇门	4,500	247	1	0.030	0.070	0.013
30扇门	12,500	247	1	0.037	0.070	0.030
100扇门	40,500	247	1	0.031	0.084	0.076
300扇门	120,500	247	1	0.030	0.106	0.180
1,000扇门	400,500	247	1	0.024	0.177	0.583
锁						
100 把锁	60,100	12,700	100	0.151	0.161	0.357
300 把锁	180,100	38,100	300	0.412	0.432	1.070
500 把锁	300,100	63,500	500	0.590	0.613	1.783
1,000 把锁	600,100	127,000	1,000	1.230	1.270	3.564
总统座驾（n–D：n–维）						
4–D	3,800	247	1	0.061	0.063	0.956
16–D	13,400	247	1	0.060	0.064	1.269
64–D	51,800	247	1	0.063	0.069	2.535
256–D	205,400	247	1	0.054	0.071	7.295
1,024–D	819,800	247	1	0.091	0.176	25.592
血糖						
$\square_{[600,700]}(x \le 200)$	2	0	8	0.000	0.078	0.002

体现了我们安全多方计算操作的高效性，其瓶颈在于输入/输出和网络协调，而非计算。

由于锁状态检查涉及大量布尔型操作，锁系统的速度比 ACS 慢，但仍保持次线性扩展：锁数量增加 $10\times$ ，延迟仅增加 $7.9\times$ 。计算时间随系统规模增长，但仍在实际可接受范围内，介于 0.151 秒至 1.230 秒之间。

在总统专车场景中，除最高维度的情况（1,024-D）下通信开销变得更加显著外，不同维度下的计算时间保持稳定（0.054–0.091 秒）。尽管计算维度很高，即使在 1,024 维时，每次迭代的时间仍远低于 200 毫秒。

血糖场景的执行速度极快，计算时间可以忽略不计，每次迭代的总时间为 0.078 秒。

与先前工作的比较

性能提升。 Henzinger 等人此前的工作 [16] 采用混淆电路，并使用工业标准安全参数 1536，报告的执行时间为至少每次迭代 18 秒，适用于可比的规范。由于

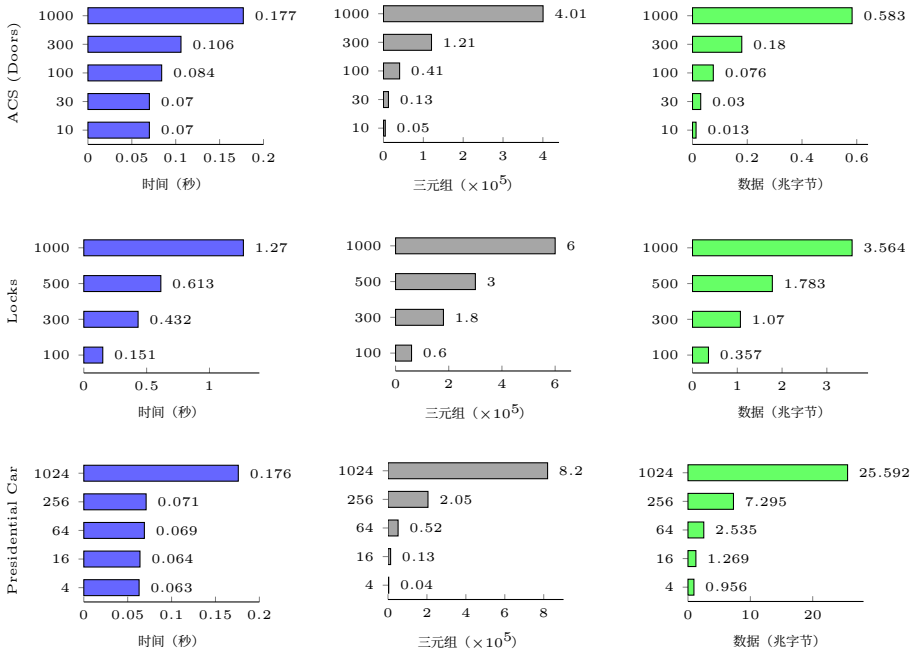

图 2：所有案例研究的性能对比：（左）每次迭代的耗时，（中）电路复杂度（三元组数量），（右）通信开销。三行分别展示 ACS、锁和总统专车场景。

规范复杂度增加时，其开销会增至每次迭代数分钟。Banno 等人的方法 [1]限制更大；正如 Henzinger 等人所指出的，由于表示这些基准测试所需的确定性有限自动机规模过大，该方法无法处理 ACS 和锁等基准测试。同样，由于类似原因，该方法也无法处理地理围栏（总统座驾）场景。对于血糖基准测试——Banno 等人的方法可以处理这类测试——我们的实现达到了他们所考虑的两种协议中较快的一种的性能，每次迭代仅需不到一秒。

相比之下，我们的方法实现了：

– **ACS：** 每次迭代 0.07–0.18 秒（快 100–250× 倍）– **锁：** 每次迭代 0.16–1.3 秒（快 14–112× 倍）– **总统座驾：**每次迭代 0.06–0.18 秒– **血糖：** 每次迭代 0.08 秒

这使执行时间提升了*2–3* 个数量级，同时保持信息论安全保证，而这些保证不依赖计算困难性假设。

可扩展性特征。图2展示了案例研究的性能特征。对于 ACS 和 Presidentialcar，耗时几乎保持不变

而预处理需求和通信量则随规模线性增长。对于锁系统，由于锁状态检查具有大量布尔型操作，所有指标都会随系统规模增长。尽管如此，我们的 Shamir 秘密共享方法仍保持了实用的性能，速度比密码学方法快 2–3 个数量级，使大规模系统的实时监控切实可行。

6 讨论

我们提出了一种隐私保护监控协议，在可扩展性方面显著优于现有方案。这种效率得益于架构化信任分布：我们用轻量级秘密共享替代了繁重的密码学原语，但这要求作出诚实多数假设（即至少有一个监控方未遭到攻破）。这体现了一种根本性的权衡：我们接受更强的架构假设，以实现实时监控所需的性能。

处理大型系统状态。当系统拥有庞大的内部状态（例如大型数据库）时，会产生另一项挑战。在这种情况下，共享整个状态并不现实。一种可能的解决方案是为我们的协议加入私有信息检索（PIR）。在此设置中，监控器充当 PIR 客户端，在不泄露其访问哪些索引的情况下检索特定状态份额；而系统则充当 PIR 服务器 [6]。这一扩展需要使用第3.1 节所介绍原语之外的额外密码学机制，是未来研究的一个有前景的方向。

与现有工具集成。 尽管我们的原型验证了核心协议的可行性，但我们尚未将这一配置集成到成熟的运行时验证框架中。未来的工作将聚焦于将我们的 MP-SPDZ 实现与 BeepBeep [4], DejaVu [15],或 MonPoly [2], 等标准监控工具对接，从而支持更为熟悉的规范语言。

抵御主动对手。 尽管我们的主要评估聚焦于半诚实设定，但该协议可以加固，以抵御可能偏离规范的主动（恶意）对手。为防止被攻陷的参与方根据其他参与方的消息调整自身份额，从而发起自适应攻击，我们可以强制要求输入承诺，即要求参与方在公开份额之前，以密码学方式将自身与这些份额绑定。为检测存储值是否遭到未授权修改，可以将该协议扩展为使用信息论消息认证码（MACs）的认证秘密共享。不过，这些技术会带来额外的通信和计算开销。SPDZ [12] 和 MASCOT [19] 等框架基于这些原理提供了高效的恶意安全协议。

致谢。 本研究是 VAMOS 项目的一部分，该项目获得了欧洲研究委员会（ERC）资助，资助协议编号为 101020093；此外，本研究还获得了奥地利科学基金（FWF）SFB 项目 SpyCoDe F8502 的支持。

参考文献

1. Banno, R., Matsuoka, K., Matsumoto, N., Bian, S., Waga, M., Suenaga, K.：通过全同态加密实现面向安全性LTL规范的不经意在线监控。载于：计算机辅助验证 - CAV。第447–468页。施普林格国际出版（2022年）2. Basin, D.A., Klaedtke, F., Zalinescu, E.：monpoly监控工具。载于：RV- CuBES（2017年33336626 3），https://api.semanticscholar.org/CorpusID:。Beaver, D.：使用电路随机化的高效多方协议。载于：密码学进展 - CRYPTO '91，第11届年度国际密码学会议，美国加利福尼亚州圣巴巴拉，1991年8月11日至15日，会议论文集。《计算机科学讲义》，第576卷，第420–432页。施普林格（1991年），Beaver乘法三元组 4. Boussaha, M.R., Khoury, R., Hallé, S.：使用BeepBeep监控安全属性。载于：Imine, A., Fernandez, J.M., Marion, J.Y., Logrippo, L., Garcia- Alfaro, J.（编）安全基础与实践。第160–169页。施普林格国际出版，查姆（2018年）5. Catrina, O., de Hoogh, S.：用于安全多方整数计算的改进原语。载于：网络安全与密码学，第7届国际会议，SCN 2010，意大利阿马尔菲，2010年9月13日至15日。会议论文集。《计算机科学讲义》，第6280卷，第182–199页。施普林格（2010年），高效比较和比特分解 6. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.：私有信息检索。J. ACM 45(6)，965–981（1998年11月）。https://doi.org/10.1145/293347.293350，https://doi.org/10.1145/293347.2933507. Cramer, R., Damgård, I., Escudero, D., Scholl, P., Xing, C.：SPD$\mathbb{Z}_{2^k}$：面向不诚实多数的高效模 2^k 安全多方计算。载于：密码学进展 - CRYPTO 2018，第二部分。《计算机科学讲义》，第10992卷，第769–798页。施普林格（2018年），高效跨域转换 8. Cramer, R., Damgård, I., Maurer, U.M.：基于任意线性秘密共享方案的通用安全多方计算。载于：密码学进展 - EUROCRYPT 2000，密码技术理论与应用国际会议，比利时布鲁日，2000年5月14日至18日，会议论文集。《计算机科学讲义》，第1807卷，第316–334页。施普林格（2000年），基于线性秘密共享的安全多方计算 9. Dalskov, A., Escudero, D., Keller, M.：量化神经网络的安全求值。载于：隐私增强技术会议论文集，PETS。第2020卷，第355–375页（2020年），利用预处理的高效比特分解 10. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.：用于等于、比较、比特和幂运算的无条件安全常数轮多方计算。载于：密码学理论，第三届密码学理论会议，TCC 2006，美国纽约州纽约，2006年3月4日至7日，会议论文集。《计算机科学讲义》，第3876卷，第285–304页。施普林格（2006年），常数轮等于和比较协议

11. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.：面向不诚实多数的实用隐蔽安全多方计算——或者说：突破SPDZ的限制。载于：计算机安全 - ESORICS 2013 - 第18届欧洲计算机安全研究研讨会，英国伊格姆，2013年9月9日至13日。会议论文集。《计算机科学讲义》，第8042卷，第1–18页。施普林格（2013年），具有隐蔽安全性的SPDZ 12. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.：基于部分同态加密的多方计算。载于：密码学进展 - CRYPTO 2012 - 第32 23届年度密码学会议，美国加利福尼亚州圣巴巴拉，2012年8月 - 日。会议论文集。《计算机科学讲义》，第19 7417卷，第643–662页。施普林格（2012年），原始SPDZ论文 13. Demmler, D., Schneider, T., Zohner, M.：ABY——用于高效混合协议安全双方计算的框架。载于：第22届年度网络与分布式系统安全研讨会，NDSS 2015，美国加利福尼亚州圣迭戈，2015年2月8日至11日。互联网协会（2015年），具有高效转换的混合算术-布尔-Yao协议 14. Goldreich, O., Micali, S., Wigderson, A.：如何进行任何心理游戏，或具有诚实多数协议的完备性定理。SIAM计算杂志**19**(1)，218–229（1990年），安全多方计算的奠基性工作 15. Havelund, K., Peled, D., Ulus, D.：DejaVu：一阶时序逻辑监控工具。载于：2018 IEEE信息物理系统监控与测试研讨会（MT-CPS）。第12–13页（2018年）。https://doi.org/10.1109/MT-CPS.2018.00013 16. Henzinger, T.A., Karimi, M., Thejaswini, K.S.：隐私保护运行时验证。载于：2025年ACM SIGSAC计算机与通信安全会议论文集。第2774–2787页。CCS '25，计算机协会，美国纽约州纽约（2025年）。https://doi.org/10.1145/3719027.3765137，https://doi.org/10376513717114537190 27././。Keller, M.：MP-SPDZ：多方计算的通用框架。载于：2020年ACM SIGSAC计算机与通信安全会议论文集，CCS 2020，线上会议，美国，2020年11月9日至13日。第1575–1590页。ACM（2020年），MP-SPDZ框架与实现 18. Keller, M.：MP-SPDZ：多方计算的通用框架。https://github.com/data61/MP-SPDZ （2024年），软件代码库 19. Keller, M., Orsini, E., Scholl, P.：MASCOT：通过不经意传输实现更快的恶意安全算术计算。载于：2016年ACM SIGSAC计算机与通信安全会议论文集，CCS 2016，奥地利维也纳，2016年10月24日至28日。第830–842页。ACM（2016年），用于恶意安全的mASCOT协议 20. Keller, M., Pastro, V., Rotaru, D.：Overdrive：让SPDZ再次强大。载于：密码学进展 - EUROCRYPT 2018 - 第37届密码技术理论与应用国际会议，以色列特拉维夫，2018年4月29日至5月3日，会议论文集，第三部分。《计算机科学讲义》，第10822卷，第158–189页。施普林格（2018年），SPDZ的改进预处理 21. Koll, C., Hang, P.T., Rosulek, M., Abbas, H.：暗中监控：信息物理系统的隐私保护运行时验证（2025年），https://arxiv.org/abs/250516059 22。Lindell, Y.：如何模拟它——模拟证明技术教程，第277–346页。施普林格国际出版，查姆（2017年）。https://doi.org/10.1007/978-3-319-57048-8_6，https://doi.org/10.1007/978-3-319-57048-8_6

23. Luo, N., Antonopoulos, T., Harris, W.R., Piskac, R., Tromer, E., Wang, X.：以零知识证明不可满足。在：2022年ACM SIGSAC计算机与通信安全会议论文集，CCS。第2203–2217页。ACM（2022）。https://doi.org/103559373 2411453548606./.. Luo, N., Judson, S., Antonopoulos, T., Piskac, R., Wang, X.：ppsat：迈向双方私有SAT求解。在：Butler, K.R.B., Thomas, K.（编）第31届USENIX安全研讨会，USENIX安全2022，美国马萨诸塞州波士顿，2022年8月10日至12日。第2983–3000页。USENIX协会（2022），https://www.usenix.org/conference/usenixsecurity22/presentation/luo 25. Luo, N., Weng, C., Singh, J., Tan, G., Raykova, M., Piskac, R.：基于TNFA的隐私保护正则表达式匹配。在：计算机安全——ESORICS 2024——第29届欧洲计算机安全研究研讨会。《计算机科学讲义》，第14983卷，第225–246页。施普林格（2024）。https://doi.org/10.1007/978-3-031-70890-9_1226. Mohassel, P., Rindal, P.：ABY3：用于机器学习的混合协议框架。在：2018年ACM SIGSAC计算机与通信安全会议论文集，CCS 2018，加拿大安大略省多伦多，2018年10月15日至19日。第35–52页。ACM（2018），采用复制秘密共享的三方计算 27. Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.：Cryptflow2：实用的双方安全推理。在：ACM SIGSAC计算机与通信安全会议，CCS。第325–342页。ACM（2020），高效的常数轮算术到布尔转换 28. Shamir, A.：如何共享秘密。《ACM通讯》**22**（11），612–613（1979年），多项式秘密共享 29. Waga, M., Matsuoka, K., Suwa, T., Matsumoto, N., Banno, R., Bian, S., Suenaga, K.：通过全同态加密对离散时间STL进行无知监测。在：运行时验证——第24届国际会议，RV。《计算机科学讲义》，第15191卷，第59–69页。施普林格（2024）。https://doi.org/10.1007/978-3-031-74234-7_430. Yao, A.C.C.：如何生成和交换秘密。在：第27届计算机科学基础年度研讨会，FOCS。第162–167页。IEEE计算机学会（1986年），用于安全双方计算的混淆电路

A 共享系统的实例化

A.1 实例化

现在我们给出该抽象模型的若干具体实例。这些示例表明，该抽象涵盖了广泛的实用秘密共享方案，且各方案在计算与安全权衡方面各不相同。

环上的加法共享。设 R 为有限环（例如，素数p对应的 $\mathbb{Z}_{2^{64}}$ 或 $\mathbb{Z}_p$ ）。定义 $\mathcal{V} = \mathcal{D} = R$。通过均匀随机选择 $v^{(1)}, \ldots, v^{(k-1)} \xleftarrow{\$} R$，并令，即可共享值 $v \in R$

$$v^{(k)} = v - \sum_{i=1}^{k-1} v^{(i)} \pmod{R}.$$

重构通过对 R 取模求和实现：

$$\text{Reconstruct}(v^{(1)}, \ldots, v^{(k)}) = \sum_{i=1}^{k} v^{(i)} \pmod{R}.$$

该方案支持本地执行加法和标量乘法（无需交互），并通过 Beaver 乘法等交互式协议执行乘法 [3]。运算集合 $\mathcal{O}$ 包括环运算 $\{+, -, \times\}$，以及通过转换为布尔形式实现的比较 [5,10]。

多项式（*Shamir*）共享。令 $\mathbb{F}$ 为有限域，并固定阈值 $t < k$。定义 $\mathcal{V} = \mathcal{D} = \mathbb{F}$。通过抽取一个次数至多为 t 的随机多项式$p(x) \in \mathbb{F}[x]$，使其满足 $p(0) = v$，并分发份额$p(\alpha_1) \ldots, p(\alpha_k)$，即可共享一个值 $v \in \mathbb{F}$。其中，$\alpha_1, \ldots, \alpha_k \in \mathbb{F}$ 是预先固定的互不相同的非零求值点。在 $x = 0$ 处进行拉格朗日多项式插值即可重构。该方案可针对最多由 t 台计算机组成的任意合谋方提供信息论隐私，并支持通过 Shamir 协议 [28]实现加法和乘法。

布尔秘密共享。令 $\mathcal{V} = \{0, 1\}$ 和 $\mathcal{D} = \mathbb{Z}_2$。比特 $b \in \{0, 1\}$ 以模 2 加法的方式共享：计算机持有随机比特 $b^{(1)}, \ldots, b^{(k)}$，使得 $b = b^{(1)} \oplus \cdots \oplus b^{(k)}$。此实例支持本地执行异或运算，并通过混淆电路 [30]或其他交互式协议实现与运算。它特别适用于布尔电路、分支和谓词求值。

份额转换的效率。

定义 4（高效份额转换）。份额转换 $\text{Convert}^{\mathcal{S}_1 \to \mathcal{S}_2}$称为高效，如果它可以通过满足以下条件的协议来实现：

1. 运行的交互轮数为常数，且独立于待转换的值、计算机的数量和安全参数；

2. 总通信复杂度以 $\log |\mathcal{V}_1 \cap \mathcal{V}_2|$、计算机的数量 k和安全参数 λ 的多项式为上界。

直观地说，高效意味着份额转换可以作为每轮协议中的构建模块使用，而不会在渐近意义上主导整体计算的成本，也不需要重构底层秘密。

份额转换示例。典型的高效转换包括：

– 算术到布尔转换（比特分解）：将共享整数 $[\![x]\!]$ 模 2^n 分解为 n 个共享比特 $[\![b_0]\!]$、……、$[\![b_{n-1}]\!]$，使得 $x = \sum_{i=0}^{n-1} b_i \cdot 2^i$。这支持按位运算和比较。现代协议 [27,9] 利用预计算的相关随机性实现常数轮复杂度，每次转换的总通信量为 $O(n)$个域元素。协议详情请参见附录 A.2。– 布尔到算术转换：将共享比特 $[\![b_0]\!]$、……、$[\![b_{n-1}]\!]$ 嵌入算术份额 $[\![x]\!] = [\![\sum_i b_i \cdot 2^i]\!]$ 中。借助并行前缀网络 [13],，可以在常数轮内以 $O(n\log\ n)$的通信量完成；或者以 $O(n)$的通信量和 $O(\log n)$轮次完成。这使得能够对布尔值执行算术运算。– 跨域算术转换：在不同算术域之间进行转换，例如，对于素数 p，从 $\mathbb{Z}_{2^k}$ 转换为 $\mathbb{Z}_p$，或在加法共享和 Shamir 共享之间进行转换。这些转换通常每个值需要 $O(1)$轮次和 $O(\log |\mathcal{V}|)$的通信量 [7]。

上述所有转换均可根据定义 4高效实现，并满足定义2中的正确性和隐私性要求。参见 [13,26] 中的代表性构造。SPDZ [12,11],Overdrive [20], 和 MP-SPDZ [17]等现代框架提供了这些原语的高效实现。

A.2 非线性运算

虽然加法和标量乘法可以在加法份额上本地执行，但比较、相等性测试和乘法等非线性运算需要交互，并且通常受益于在不同秘密共享表示之间进行转换。

通过比特分解进行比较。为了计算 $[\![x]\!] < [\![y]\!]$，其中 $x,\ y \in \mathbb{Z}_{2^n}$是算术秘密共享值，请注意，关系 $x < y$ 不会在随机份额相加后保持（即，$x^{(i)} < y^{(i)}$ 不意味着 $x < y$）。一种方法 [5,10] 是转换为布尔表示：

1. **比特分解：** 将两个算术共享转换为布尔形式。对于共享值 $[\![x]\!]$，生成共享比特 $[\![b_0]\!], \ldots, [\![b_{n-1}]\!]$（每个都是 $\mathbb{Z}_2$ 中的布尔共享），使得 $x = \sum_{i=0}^{n-1} b_i \cdot 2^i$。

现代协议只需常数轮即可完成此操作，每个值的通信量为 $O(n)$ [27,9]。其核心思路是在离线阶段预先计算相关随机性：生成随机布尔共享 $[\![r_0]\!], \ldots, [\![r_{n-1}]\!]$，以及它们的算术和 $[\![r]\!] = [\![\sum_i r_i \cdot 2^i]\!]$。

在在线阶段，计算并公开 $c = x + r \bmod 2^n$。由于 r 是均匀随机的，c 不会泄露关于 x的任何信息。随后，计算机在本地计算 c_0、.....、c_{n-1} 的公开比特，并使用预先计算的随机性，通过常数轮协议计算 $[\![b_i]\!] = c_i - [\![r_i]\!] - [\![\text{进位}_{i-1}]\!]$，其中 $[\![\text{进位}_{i-1}]\!]$ 表示二进制加法 $x + r$中位置 $i-1$ 产生的进位比特。

早期基于顺序进位传播的协议 [5]需要 $O(n)$ 轮次，但如今已被常数轮构造取代。

2.**按位比较：**当 x 和 y 均采用布尔形式时，求值一个布尔电路，从最高有效位开始逐位计算比较结果。该电路的深度为 $O(\log n)$，规模为 $O(n)$，因此需要 $O(\log n)$ 轮次，通信量为 $O(n)$ [10]。电路输出一个布尔共享 $[\![x < y]\!]$。

3. **可选的反向转换：**如果结果需要采用算术形式（例如用于条件操作），则通过布尔型到算术型转换，在 $O(1)$ 轮次内、以 $O(1)$ 通信量将布尔共享转换为算术份额。

总体复杂度：每次比较总共需要 $O(\log n)$ 轮次和 $O(n)$ 通信量，其中 n 是待比较数值的位长度。

相等性测试。相等性测试 $[\![x]\!] \stackrel{?}{=} [\![y]\!]$ 比一般比较更高效。进行比特分解后（这是主要开销，需要 $O(1)$ 轮次和 $O(n)$ 通信量），计算对应比特的逐位异或：$[\![d_i]\!] = [\![x_i]\!] \oplus [\![y_i]\!]$ 对 $i = 0$，.....，$n-1$。这是本地操作（无需通信）。然后，当且仅当所有 $d_i = 0$ 均满足条件时，$x = y$；这可以通过计算或树来测试：$[\![d_0 \vee d_1 \vee \cdots \vee d_{n-1}]\!]$。使用由或门构成的二叉树，需要 $O(\log n)$ 轮次和 $O(n)$ 次乘法（ $\mathbb{Z}_2$ 个与门可根据德摩根定律实现或门）。如果 $x = y$，输出为 0（真）；否则输出为 1（假）。

该方法与比较运算具有相同的渐近复杂度（$O(\log n)$ 轮次，$O(n)$ 通信），但由于避免了完整的比较电路，其常数更小。

算术域中的乘法。加法共享或 Shamir 共享中的乘法需要交互。标准方法使用 *Beaver* 三元组 [3]：预先计算的共享值 $[\![a]\!]$，$[\![b]\!]$，$[\![c]\!]$ 其中 $c = a \times b$和 a，b 均匀随机。给定这样一个三元组，计算机在一轮中公开 $[\![e]\!] = [\![x - a]\!]$ 和 $[\![f]\!] = [\![y - b]\!]$，然后在本地计算

$$[\![z]\!] = [\![c]\!] + e \cdot [\![b]\!] + f \cdot [\![a]\!] + e \cdot f.$$

由于 e 和 f 被随机值掩盖，因此不会泄露任何关于 x 或y的信息。这需要 $O(1)$轮次，并且每次乘法需要 $O(1)$ 个域元素的通信（假设 Beaver 三元组已在离线阶段预先计算）[12]。

B 我们协议的操作语义

类型化寄存器。 监控器维护一组有限的抽象存储位置 R_0， R_1，……，R_{n-1}，我们类比机器指令集，将其称为寄存器 $\mathcal{S}$。每个寄存器都是一个逻辑存储单元，用于按照约定的共享系统保存共享值。具体而言，寄存器 R_i 表示一个份额元组（$s_i^{(1)}$，……，$s_i^{(k)}$），分布在各台监控计算机上，其中计算机 j 存储份额 $s_i^{(j)}$ 。这些并非物理 CPU 寄存器，而是分布式计算中的抽象内存位置。

每个寄存器都有一个秘密共享类型，用于指明当前采用的表示形式：

– **算术型**（Arith）：寄存器保存环／域元素的加法秘密共享或多项式秘密共享。原生支持操作 $\{+,-,\times\}$。– **布尔型**（Bool）：寄存器保存布尔秘密共享（基于异或）。原生支持操作 $\{\wedge,\vee,\oplus,\neg\}$。

我们用 R_i：算术型表示寄存器 R_i 当前持有算术共享；布尔型的情况也类似。

在第 t 轮开始时，规范状态 $[\![\mu_t]\!]$ 会载入指定的一组寄存器中，而新近共享的可观测输出 $[\![x_t]\!]$ 则载入其他寄存器中。所有寄存器都会根据规范的要求，以一致的秘密共享类型初始化。

指令集。监控器通过执行一系列类型化指令来评估规范。每条指令都对共享值进行操作，而不会重构这些值，唯有最终的违规标志例外。

指令集包括：

– **算术运算**（所有操作数和结果都必须为算术型）：

ADD R_i， R_j， R_k 计算 $[\![R_i]\!] \leftarrow [\![R_j + R_k]\!]$SUB R_i， R_j，R_k 计算 $[\![R_i]\!] \leftarrow [\![R_j - R_k]\!]$MUL R_i， R_j， R_k 计算 $[\![R_i]\!] \leftarrow [\![R_j \times R_k]\!]$ADDC R_i， R_j， c 计算 $[\![R_i]\!] \leftarrow [\![R_j + c]\!]$ 针对公开值 c

– **布尔运算**（所有操作数和结果都必须是布尔型）：

异或 R_i, R_j, R_k 计算 $[\![R_i]\!] \leftarrow [\![R_j \oplus R_k]\!]$与 R_i, R_j, R_k 计算 $[\![R_i]\!] \leftarrow [\![R_j \wedge R_k]\!]$非 R_i, R_j 计算 $[\![R_i]\!] \leftarrow [\![\neg R_j]\!]$

– **比较操作** （操作数必须为算术型，结果为布尔型）：

$$\begin{aligned} &\texttt{LT}\ R_i, R_j, R_k && \text{compute } [\![R_i]\!]_{\texttt{Bool}} \leftarrow [\![R_j < R_k]\!] \\ &\texttt{EQ}\ R_i, R_j, R_k && \text{compute } [\![R_i]\!]_{\texttt{Bool}} \leftarrow [\![R_j = R_k]\!] \end{aligned}$$

份额转换就在此处发生。 在内部，比较操作会执行份额转换，将其转换为适用于比较的中间表示形式。具体转换取决于底层的安全多方计算协议：它可能涉及将算术共享转换为布尔共享（例如，将整数分解为共享比特以进行按位比较），也可能涉及不同算术表示形式之间的转换（例如，从加法共享转换为多项式共享）。比较协议对这些转换后的份额进行运算，并生成一个布尔共享的结果比特 $[\![R_i]\!]_{\texttt{Bool}}$。

转换协议 $\text{Convert}^{\mathcal{S}_{\texttt{Arith}} \to \mathcal{S}_{\texttt{Bool}}}$ （或不同算术表示形式之间的转换），在第3.2节中介绍，会由这些比较指令在内部调用。

– **重构** （仅用于最终的违规标志）：

$$\texttt{REVEAL}\ b, R_i \qquad\qquad \text{reconstruct } b \leftarrow \text{Reconstruct}([\![R_i]\!])$$

此处的 b 是所有监控计算机均可见的公开输出比特。这是唯一 会揭示秘密值的指令。

类型规则。 每条指令都有一个类型签名，用于约束其操作数和结果允许使用的共享类型。若每条指令所作用的寄存器当前共享类型均符合该指令的要求，则指令序列类型正确 。

规范编译器（将高级规范（ϕ， δ）转换为指令序列）必须确保生成的指令序列具有良好类型。在此协议中，份额转换仅发生在比较操作（LT、等于）中，这些操作会在内部将算术共享操作数转换为适用于比较协议的中间形式。

这使份额转换的成本明确且可度量：通过计算比较操作的数量，我们可以精确核算协议的转换开销。

示例：计算 $\phi(\mu_t,\ x_t) = (x_t > \mu_t)$。假设 $[\![\mu_t]\!]$ 属于 R_0 ： 算术型，且$[\![x_t]\!]$ 属于 R_1 ：算术型。指令序列如下：

$$\begin{aligned} &\texttt{LT}\ R_2, R_0, R_1 && \text{compute } [\![\mu_t < x_t]\!] \to R_2 : \texttt{Bool} \\ &\texttt{REVEAL}\ \mathit{flag}, R_2 && \text{output the violation flag} \end{aligned}$$

LT 指令会在内部执行必要的份额转换：将 $[\![\mu_t]\!]_{\texttt{Arith}}$ 和 $[\![x_t]\!]_{\texttt{Arith}}$ 转换为比较协议所需的中间表示，执行比较，并在 R_2中输出一个布尔共享结果比特。

示例：状态更新 $\mu_{t+1} = \mu_t + x_t$ 模256。 假设规范维护一个模256的累加和。初始时， $[\![\mu_t]\!]$ 的类型为 R_0：算术型， $[\![x_t]\!]$ 的类型为R_1：算术型：

$$\texttt{ADD}\ R_3, R_0, R_1 \qquad\qquad [\![\mu_{t+1}]\!] \leftarrow [\![\mu_t + x_t]\!]$$

由于所有值都是算术共享的，因此无需进行份额转换。寄存器 R_3 保存 $[\![\mu_{t+1}]\!]$，供下一轮使用。

示例：复杂谓词 $(x_t > 100) \wedge (\mu_t = 5)$。 这需要进行比较并使用布尔逻辑：

小于 R_4, 100, R_1	$[\![100 < x_t]\!] \to R_4$： 布尔型
等于 R_5, R_0, 5	$[\![\mu_t = 5]\!] \to R_5$： 布尔型
与 R_6, R_4, R_5	$[\![R_4 \wedge R_5]\!] \to R_6$： 布尔型
公开 标志 R_6	

每个比较（小于和等于）都会对其算术共享操作数在内部执行必要的份额转换。随后，与操作会在无需任何转换的情况下合并布尔结果，因为两个输入都已经是布尔共享。

关于实际实现的备注。 现代安全多方计算框架，例如 MP- SPDZ [17] 采用了类似的基于指令的模型，其中类型化存储和份额转换嵌入比较操作之中。我们的形式化描述在协议描述中明确呈现了这一计算基础，从而能够精确推理规范评估的成本和正确性。特别是，通过确定比较是份额转换成本的唯一来源，我们可以优化规范，以尽量减少每轮执行的比较次数。

C 组合性与安全性

定理 1（跨轮次组合性）。无论协议重复执行多少次，隐私性都能得到保留。

证明。根据性质4，第 t 轮唯一泄露的信息是单个比特$\phi(\mu_t, x_t)$。根据性质2（共享系统隐私性），当任何监控器计算机的真子集查看时， $[\![\mu_{t+1}]\!]$的份额与 μ_{t+1} 相互独立。因此，该协议可以无限期地继续执行，且除了违规标志序列之外，不会累积额外的信息泄漏。

定理 2（系统隐私）。对于任何满足 $|I_M| < k_M$ 的对手结构(I_M)，受攻击的监控计算机的视图不会泄露有关系统内部状态 σ_t 或可观测输出 x_t 的任何信息，除非这些信息可由重构的违规标志序列 $\phi(\mu_1, x_1) \ldots, \phi(\mu_T, x_T)$推知。

证明。固定一个满足 $|I_M| < k_M$ 的对手结构 (I_M)。在每一轮 t，系统都会向监控计算机发送 x_t 的份额。根据性质 2（共享系统隐私性），受攻击的监控计算机所收到的份额 $\{\text{Share}(x_t)_i\}_{i\in I_M}$ 的联合分布独立于 x_t。形式上，对于任意两个可能的输出 x_t，$x'_t \in \mathcal{V}$，受攻击计算机所持份额的分布完全相同，因此不会泄露实际共享的是哪个值。

每轮唯一重构的值是标志 $\phi(\mu_t, x_t)$。这种重构会泄露关于 x_t 的一比特信息，即二元组 (μ_t, x_t) 是否满足违规条件。不会泄露关于 σ_t 的其他信息（其中包含 x_t，也可能包含其他私有组件）。

由于监控计算机使用能够保持性质2的安全协议，对共享值执行所有中间计算，因此中间结果不会泄露信息。根据轮次进行归纳可知，泄露给受攻击监控计算机的全部信息恰好是标志序列，不多不少。

定理 3（监控器隐私）。对于任意对手结构 (I_M)，且满足 $|I_M| <k_M$，受攻击的系统计算机的视图不会泄露关于监控器的规范状态 μ_t、下一状态函数 δ 或标志函数 ϕ的任何信息，除非这些信息可由重构的违规标志序列 $\phi(\mu_1, x_1)$……，$\phi(\mu_T, x_T)$ 推知。

证明（定理 3的证明）。系统实体不参与监控器的内部计算。系统的职责仅限于：（1）向监控计算机发送 x_t 的份额，以及（2）接收重构后的标志值$\phi(\mu_t, x_t)$，该操作在每轮结束时进行。

系统发送的份额由系统自身独立生成，不会泄露监控器将如何处理这些份额。监控器绝不会向系统发送份额或中间值。从监控器流向系统的唯一信息是单个比特 $\phi(\mu_t, x_t)$；无论如何，系统都会获知该信息（根据监控任务的定义）。

由于 μ_t 由监控实体以共享形式维护，且从未被重构（标志隐式体现的情况除外），并且被攻陷的系统计算机不属于监控实体，因此它们不会收到 μ_t 的任何份额，也无法获知其内容。同样，函数 δ 和 ϕ 由监控器在内部通过安全多方计算求值，除输出比特外，不会向系统泄露有关其结构的任何信息。

D 响应式监控模板

我们的实现采用反应式监控模板，在每一轮接收系统输入、更新内部状态、检查是否存在违规，并报告故障。以下代码展示了核心结构：

列表 1.1：反应式监控系统模板

```
from Compiler .library import *

程序 . 使用_edabit（真）

n, T  = int ( program . args [1]) , int ( program . args [2]) PORT_NUM
  = 14000

def spec_function ( 监控器_state , 系统_inputs ): next_state  =  δ( 监控器_
state , 系统_inputs ) 故障  =  φ( next_state ) final_state  =  故障 * 监控器_
state  +  (1 - 故障) * next_state return 故障 , final_state



监控器_状态  =...

监听_来自_客户端的连接（端口_号）
客户端_ID  = 接受_客户端_连接（端口_号）

对 t 属于 range（T）： 系统_输入  = sint . 从_客户端接收_（n, 客户端_ID）故障,
监控器_状态  = spec_函数（监控器_状态, 系统_输入）sint . 向_客户端公开_（[
客户端_ID ] , [故障]）


关闭客户端连接（客户端_ID）
```